

面向对数与指数函数的可重构阵列结构

吕 青, 蒋 林, 邓军勇, 李雪婷

(西安邮电大学 电子工程学院, 陕西 西安 710121)

摘 要: 对数与指数函数等超越函数应用广泛, 但传统实现需要针对每一类函数开发专用硬件, 周期长、成本高、灵活性差, 为了适应体系结构向自主运算发展的趋势, 提高超越函数处理的灵活性, 本文面向对数和指数两种函数提出了一种可重构阵列结构. 通过将两种函数处理算法向阵列结构中的对应处理元 PE(Processing Element)进行映射完成重构. 仿真结果表明, 可重构计算技术兼顾了计算的灵活性与高效性, 与专用硬件实现性能相当, 最高频率可以达到 210.117 MHz.

关键词: 对数函数; 指数函数; 可编程; 可重构; 阵列处理器

中图分类号: TP302

文献标识码: A

文章编号: 1000-7180(2016)10-0028-04

A Reconfigurable Array Structure for Logarithmic and Exponential Functions

LV Qing, JIANG Lin, DENG Jun-yong, LI Xue-ting

(School of Electronic Engineering, Xi'an University of Posts and Telecommunications, Xi'an 710121, China)

Abstract: The logarithmic and exponential functions are widely used, but the traditional implementation needs to be developed for each class of functions, which can lead to a long cycle, high cost and poor flexibility. In order to adapt to the development trend of the system structure, improve the flexibility of the transcendental function, this paper presents a reconfigurable array structure with two functions. The logarithmic and exponential functions are realized by programming, and then the reconfiguration is performed by configuring the processing element PE of the array structure. Based on the simulation results, reconfigurable computing technology takes into account the flexibility and efficiency, and it has the same performance with the application specific circuit, the maximum frequency up to 210 MHz.

Key words: logarithmic function; exponential function; programmable; reconfigurable; array processor

1 引言

对数与指数函数在视频处理、粒子滤波、RBF神经网络、4G 系统等领域^[1]应用广泛. 传统的对数与指数函数的实现以硬件为主^[2], 尽管功耗低、速度快, 但是是一种结构只能完成一个函数的计算, 不可重编程, 灵活性差.

近年来, 可重构计算技术逐渐成为高性能计算机系统研究中的新热点, 传统意义上的硬件和软件界限变得模糊. 可重构计算使硬件可“编译”, 在可编程的硬件介质中提供更大的计算能力和密度, 并能

在单片系统上完成各种各样的新任务, 大幅度提高芯片系统的总体性能, 实现片上超级计算^[3].

本文在此技术背景下, 提出了一种可重构的阵列结构, 使用 4×4 的处理元 PE(Processing Element)阵列结构, 以编程的方式实现对数与指数函数的运算, 在不降低最大频率的条件下解决硬件实现对数和指数函数灵活性差的问题.

2 可重构阵列结构

2.1 相关研究

常见的可重构硬件模型包括 Morphosys^[4]、Pipe-

Rench^[5]和RCA^[6],虽然它们已经证明了粗粒度的可重构处理器能够有效的提高多媒体应用的执行速度,但是仍然存在着不足. Morphosys 不能很好的支持并行运算, PipeRench 只能处理并行性的流水运算, RCA 在处理较小的运算时硬件面积浪费严重.

2.2 本文提出的结构

本文所提出的可重构阵列结构,采用处理单元 PE 之间邻接互连,同时实现基于单指令多数据 (SIMD) 的数据级并行计算和多指令多数据 (MIMD) 的指令级并行计算,解决了数据并行算法和非数据并行算法的计算高效性和编程灵活性^[7].

图 1 给出了该阵列结构图,包含 1 个 4×4 的可编程可重构 PE 阵列和主处理器连接,为了提高可编程可重构阵列的灵活性,采用一个全局指令存储器(IM)和全局控制器(GLOBAL CONTROLLER)来实现指令的下发.

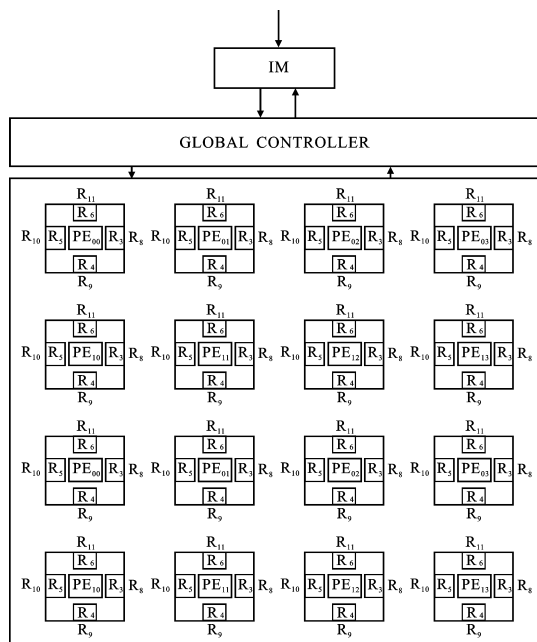


图 1 可重构阵列结构图

图 2 给出了单个 PE 的功能框图,由图可以看出 PE 由一个指令 RAM、一个数据 RAM、一个指令译码寄存器、译码单元、寄存器文件 (RegFile)、PC 控制器、运算单元和一个邻接互连控制器组成. 每个 PE 能够完成简单的算数运算(加、减等)、逻辑运算(与、或、非、跳转等).

PE 之间邻接互连,依靠共享寄存器进行通信. 如图 3 所示,当从左向右传数据时寄存器 R_3 和寄存器 R_{10} 相邻;当从上向下传数据时寄存器 R_4 和寄存器 R_{11} 相邻;当从右向左传数据时寄存器 R_5 和寄存

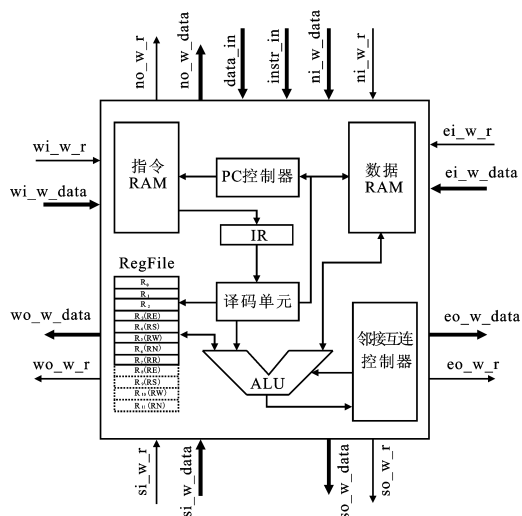


图 2 PE 的逻辑结构图

器 R_8 相邻. 当 PE_{00} 需要传递数据给 PE_{01} 时,只需要将数据写入到寄存器 R_3 中,下一个时钟周期时,寄存器 R_{10} 接收来自 R_3 的数据完成相邻处理器间的数据传递.

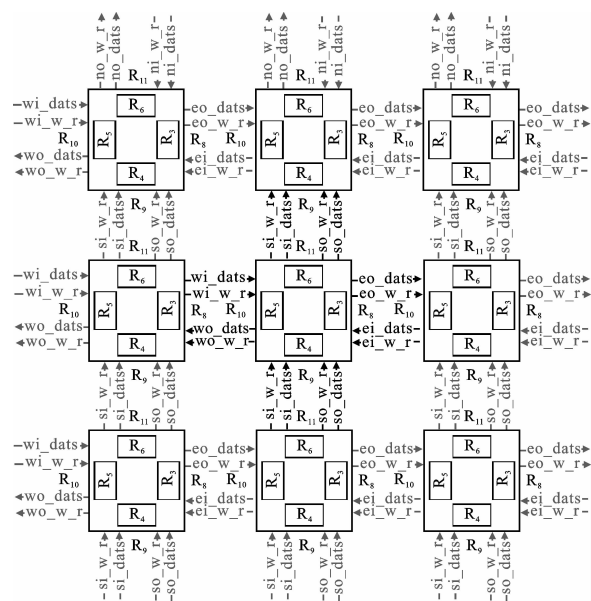


图 3 PE 的邻接互连拓朴图

3 算法的实现

由于该算法的收敛速度与数据表示的精度密切相关. 设操作数的精度为 k 位, 考虑到当 $\epsilon^2 < \text{lulp}(\text{Unit of Last Place})$ 时, $\ln(1+\epsilon) = \epsilon - \epsilon^2/2 + \epsilon^3/3 - \dots \approx \epsilon$, 即当迭代次数 $i > |k|/2$ 时, $\ln(1+d_i 2^{-i}) \approx d_i 2^{-i}$, 当 $i > k$ 时, $2^{-i} \approx 0$, 停止迭代. 考虑到, 本结构中每个 PE 的操作数采用 8 位整数、8 位小数共 16 位有符号定点数表示, 最高位为符号位, 精度为 8, 因此使

用 8 个 PE 迭代 8 次来实现 $\ln x$ 的计算。

3.1 $\ln x$ 对数函数

3.1.1 算法描述

该算法属于回退分解算法^[8](restoring decomposition algorithm),迭代公式为:

$$\begin{cases} x^{(i+1)} = x^{(i)} c^{(i)} = x^{(i)} (1 + d_i 2^{-i}) \\ y^{(i+1)} = y^{(i)} - \ln c^{(i)} = y^{(i)} - \ln(1 + d_i 2^{-i}) \end{cases}$$
$$i = 0, 1 \cdots \quad (1)$$

每一次按 $d_i = 1$ 进行迭代,若 $x^{(i)} (1 + 2^{-i}) < 1$,则 $d_i = 1$,否则 $d_i = 0$

迭代终止的条件为: $x^{(m)} = 1$
迭代终止时: $y^{(m)} \rightarrow y^{(0)} + \ln x^{(0)}$
令 $y^{(0)} = 0$,则 $y^{(m)} \rightarrow \ln x^{(0)}$

3.1.2 算法的编程实现

实现 $\ln x$ 对数函数的计算如下. 根据 3.1.1 中对该算法的描述,事先计算出 $\ln(1 + 2^{-i})$ 的值并放置在对 PE 的 1 号地址中,如表 1 所示.

表 1 存储中的数据及对应的 PE

i	$\ln(1 + 2^{-i})$	对应的 PE
0	0.693 147 18	PE ₁₀
1	0.405 465 11	PE ₁₁
2	0.223 143 55	PE ₁₂
3	0.117 793 03	PE ₁₃
4	0.060 624 62	PE ₃₃
5	0.030 771 65	PE ₃₂
6	0.015 504 18	PE ₃₁
7	0.007 782 14	PE ₃₀

具体的实现过程如图 4 所示:(1) 在 PE₀₀ 和 PE₁₀ 中完成第一级的迭代. 将初值 $x^{(0)}$ 存储在 PE₀₀ 的寄存器中,将初值 $y^{(0)}$ 存储在 PE₁₀ 的寄存器中,在 PE₀₀ 中计算并判断出 $x^{(1)}$ 的值,将判断结果存入寄存器 R₃ 中通过邻接互连传送给 PE₀₁ 的寄存器 R₁₀,并将计算出的 $x^{(1)}$ 的值传给 PE₀₁;同时在 PE₁₀ 中计算出 $y^{(1)}$ 的值,根据 PE₀₀ 中传过来的判断结果,决定 $y^{(1)}$ 是等于初始值 $y^{(0)}$ 还是等于 $y^{(1)}$, $y^{(1)}$ 计算完成后再将结果传送给 PE₁₁;(2) 在 PE₀₁ 和 PE₁₁ 中完成第二级的迭代. 当接收到 PE₀₀ 传送过来的 $x^{(1)}$ 的值时,PE₀₁ 开始计算并判断 $x^{(2)}$ 的值,完成后将判断结果传送给 PE₁₁ 并将计算结果 $x^{(2)}$ 传给 PE₀₂;同时在 PE₁₁ 中计算出 $y^{(2)}$ 的值,根据 PE₀₁ 中传过来的判断结果,决定 $y^{(2)}$ 是等于初始值 $y^{(1)}$ 还是等于 $y^{(2)}$, $y^{(2)}$ 计算完成后再将结果传送给 PE₁₂;以此类推在 16 个 PE 中完成 8 次迭代,最终在 PE₃₀ 中输出迭代的 y 的结果就是 $\ln E$ 对数函数的结果.

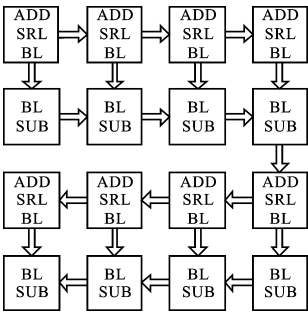


图 4 对数函数映射结构图

3.2 $\ln E$ 指数函数

3.2.1 算法描述

该算法也是一种回退分解算法^[8](restoring decomposition algorithm) 迭代公式如下:

$$\begin{cases} x^{(i+1)} = x^{(i)} c^{(i)} = x^{(i)} (1 + d_i 2^{-i}) \\ y^{(i+1)} = y^{(i)} - \ln c^{(i)} = y^{(i)} - \ln(1 + d_i 2^{-i}) \end{cases}$$
$$i = 0, 1 \cdots \quad (2)$$

每一次按 $d_i = 1$ 进行迭代,若 $y^{(i)} - \ln(1 + 2^{-i}) \geq 0$,则 $d_i = 1$,否则 $d_i = 0$

迭代终止的条件为: $y^{(m)} = 0$
迭代终止时: $x^{(m)} \rightarrow x^{(0)} e^{y^{(0)}}$
令 $x^{(0)} = 1$,则 $x^{(m)} \rightarrow e^{y^{(0)}}$

3.2.2 算法的编程实现

实现 $\ln x$ 指数函数的计算如下. 根据 3.2.1 节中对该算法的描述,事先计算出 $\ln(1 + 2^{-i})$ 的值并放置在对 PE 的 1 号地址中,如表 1 所示. 具体的实现过程如图 5 所示:(1) 在 PE₀₀ 和 PE₁₀ 中完成第一级的迭代. 将初值 $x^{(0)}$ 存储在 PE₀₀ 的寄存器,将初值 $y^{(0)}$ 存储在 PE₁₀ 的寄存器中,在 PE₁₀ 中计算并判断出 $y^{(1)}$ 的值,将判断结果存入寄存器 R₆ 中通过邻接互连传送给 PE₀₀ 的寄存器 R₉ 中并将计算出的 $y^{(1)}$ 的值传给 PE₁₁.

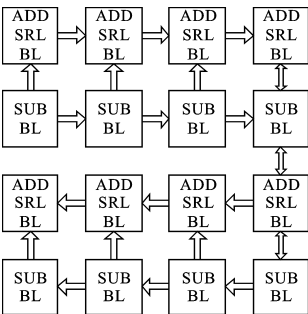


图 5 指数函数映射结构图

同时在 PE₀₀ 中计算出 $x^{(1)}$ 的值,根据 PE₀₀ 中传过来的判断结果,决定 $x^{(1)}$ 的 $x^{(0)}$ 的是等于初始值

$x^{(0)}$ 的还是等于 $x^{(1)}$ 的, $x^{(1)}$ 的计算完成后再将结果传送给 PE_{01} ; (2) 在 PE_{01} 和 PE_{11} 中完成第二级的迭代. 当接收到 PE_{10} 传送过来的 $y^{(1)}$ 的值时, PE_{11} 开始计算并判断 $y^{(2)}$ 的值, 完成后将判断结果传送给 PE_{01} 并将计算结果 $y^{(2)}$ 的传给 PE_{12} ; 同时在 PE_{01} 中计算出 $x^{(2)}$ 的值, 根据 PE_{01} 中传过来的判断结果, 决定 $x^{(2)}$ 的是等于初始值⁽¹⁾ 的还是等于 $x^{(2)}$ 的, $x^{(2)}$ 的计算完成后再将结果传送给 PE_{02} ; 以此类推在 16 个 PE 中完成 8 次迭代, 最终在 PE_{20} 中输出迭代的 x 的结果就是 $\ln x$ 指数函数的结果.

3.3 可重构的配置系统

可重构的配置系统分为可重构阵列级和处理元级.

在计算开始之前, 全局控制器接受外部给出的配置使能, 根据外部配置的是使能判断需要下发的全局指令存储器的首地址及深度, 并从全局指令存储器中读取下发.

当所有的处理元的内部指令寄存器都初始化完成后, 全局控制器选取某组外部配置信息, 启动阵列开始进行计算, 全局控制器可以控制处理元的执行, 处理元的执行是以外部配置信息位单位的, 可以循环执行外部配置信息, 以可以在多组配置信息之间进行切换. 中间结果通过寄存器输出来实现动态重构.

4 结果及分析

本结构中 PE 的操作数采用 8 位整数、8 位小数的 16 位有符号定点数表示. 图 6 是 $\ln 0.6$ 的输出结果仿真图, 输出结果为 -129, 换算为二进制为 100 000 001 000 000 1, 按照本文对于定点数的划分规定, 结果为 -0.503 906 25, 与实际结果误差为 1.3%. 图 7 是 $e^{0.6}$ 的输出结果仿真图, 输出结果为 462, 换算后为 1.832 031 25, 误差为 1.1%.

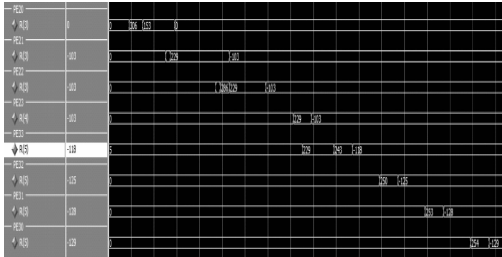


图 6 $\ln 0.6$ 的计算结果

基于 Xilinx XC6VLX550T FPGA 芯片分别对 $\ln E$ 和常规 CORDIC 算法进行综合, 性能比较如表 2 所示, 虽然收敛迭代次数的限制无法达到很高的计

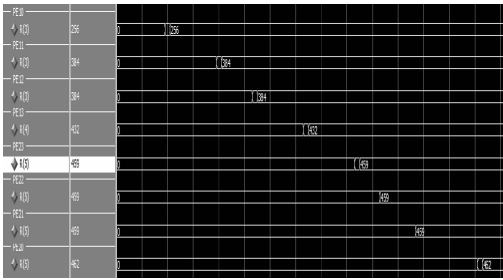


图 7 $e^{0.6}$ 的计算结果

算精度, 但是和传统的 CORDIC 算法的实现相比, 时钟频率很高, 占用资源也较少.

表 2 性能对比

	本文	CORDIC 算法
迭代级数	8	10
时钟频率	210.117 MHz	196.99 MHz
逻辑器件	1 592(4%)	17 913(6%)

5 结束语

传统实现超越函数需要针对每一类函数开发专用硬件, 周期长、成本高、灵活性差, 为了适应体系结构向自主运算发展的趋势, 并且提高超越函数处理的灵活性, 本文提出了一种可重构的阵列结构, 用可重构的方式灵活的实现了对数和指数函数的计算.

实验结果表明, 与之前文献中出现的硬件实现两种函数相比, 最大频率基本相同, 但灵活性大大提高, 更加实用.

参考文献:

[1] 张朝杰, 金小军. 基于 CORDIC 算法的微小卫星接收机设计及实现[J]. 浙江大学学报: 工学版, 2008, 42(6): 960-964.

[2] Sidahao N, Constantinides G A, Cheung P Y. Architectures for function evaluation on FPGAs[C]// Proceedings of IEEE International Symposium on Circuits and Systems, London, 2003: 804-807.

[3] Walther J S. A unified algorithm for elementary functions[C]// Proceedings of AFIPS Spring Joint Computer Conference, USA, Atlantic, 1971: 379-385.

[4] Hartej Singh, Guangming Lu. MrophoSys: An integrated reconfigurable system for date-parallel and computation intensive applications[C]// Proceedings of IEEE International Symposium on Circuits and Systems, New York, 2000: 465-481.

(下转第 35 页)

正常调度),如果小于负载保护线,说明可以进行节能管理,则通知负载均衡任务将使用率较低的 CPU 上的任务分配到该组的另一个 CPU 上,并关闭该 CPU.当任务在唤醒后,根据亲和性检查当前 CPU 是否处于关闭状态,如果处于关闭状态,则选择组亲和的另一个 CPU.

该算法的实现过程如图 4 所示:假设 CPU1 在启动后,CPU 的使用率位于负载保护线和节能线之间,经过一段时间后,CPU1 的使用率达到高负载警戒线,此时在 CPU0 上运行的负载均衡任务发现 CPU1 的使用率超过了负载保护线,则负载均衡任务将 CPU1 上的次优先级任务调度到 CPU0 上运行,并减小 CPU1 的使用率.在每次执行完一次任务调度后,负载均衡任务挂起自己,等待由统计任务发出的信号量.

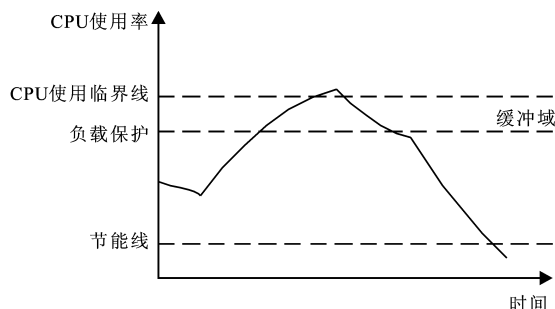


图 4 CPU 负载变化图

为了保证实时性,可以预留前两个 CPU 不参与组亲和算法,这两个 CPU 上的任务仅仅亲和对应的 CPU.可以将一些重要的高优先级任务放到这两个 CPU 中,从而及时处理中断.因为负载均衡任务对 CPU 的使用率调度是需要时间过渡的,可能导致一些重要的高优先级任务得不到及时处理.

通过上述机制,在确保实时性的基础上,实现组亲和调度算法,解决亲和调度算法中负载均衡的问题,以及在处理器利用率较低时,实现组内亲和和队列的融合,关闭某些 CPU,达到减小系统功耗的

目的.

该算法也会带来一些问题,由于每个核增加了一个统计任务,可能会增加每个核的系统负载.

5 结束语

本文针对 VxWorks 嵌入式实时多核操作系统现有亲和任务调度方法存在的多核负载不均衡问题,提出了改进的组亲和调度算法,与原来的亲和调度算法相比,理论上该方法在不破坏实时性的前提下,解决了亲和调度算法中 CPU 负载不均衡的问题,同时降低了系统功耗.后继实验验证将在下一步的工作中进行,同时也将对组亲和调度算法的分组粒度开展更加深入的研究.

参考文献:

- [1] Vaswani R, Zahorjan J. The implications of cache affinity on processor scheduling for multiprogrammed shared-memory multiprocessors [C] // Proc. 13th Symp. On Operation systems Principle. USA. Pacific Grove, ACM, 1991:26-40.
- [2] Andrew S Tanenbaum. 现代操作系统[M]. 3 版. 北京:机械工业出版社,2009.
- [3] Jean J Labrosse. 嵌入式实时操作系统 uC/OS-II[M]. 2 版. 北京:北京航空航天大学出版社,2003.
- [4] John L Hennessy, David A Patterson. 计算机体系结构:量化研究方法[M]. 5 版. 北京:机械工业出版社,2011.

作者简介:

康 鹏 男,(1991-),硕士研究生.研究方向为嵌入式操作系统. E-mail:734289208@qq.com.

刘从新 男,(1976-),博士,高级工程师.研究方向为多核并行框架.

沈绪榜 男,(1931-),研究员,博士生导师.中国科学院院士.研究方向为嵌入式计算机及其芯片.

(上接第 31 页)

- [5] Goldstein S C, Schmit H PipeRench. A reconfigurable architecture and compiler[C]// Proceedings of IEEE International Symposium on Circuits and Systems. New York, 2000: 70-77.
- [6] 肖钰,刘雷波. 应用于视频处理器的可重构流水处理器的设计与实现[J]. 电视技术,2008,32(6):18-20.
- [7] 沈绪榜,刘泽响,王茹. 计算机体系结构的统一模型[J]. 计算机学报,2007,29(5):729-736.
- [8] 牟胜梅,李兆刚. 一种面向 FPGA 的指/对数函数求

值算法[J]. 计算机工程与应用,2011,33(4):59-61.

作者简介:

吕 青 女,(1989-),硕士研究生.研究方向为集成电路系统设计. E-mail:390396308@qq.com

蒋 林 男,(1970-),教授.研究方向为专用集成电路设计.

邓军勇 男,(1981-),副教授.研究方向为专用集成电路设计.